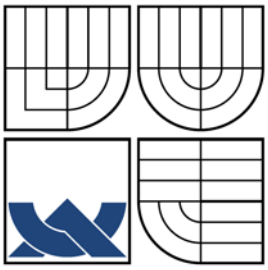
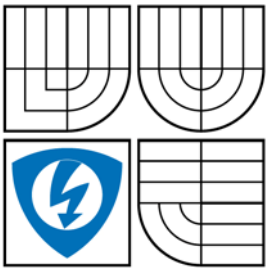


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATION

WAVELETOVÁ TRANSFORMACE V REÁLNÉM ČASE V PROSTŘEDÍ PURE DATA

REAL-TIME WAVELET TRANSFORM IN PURE DATA

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR
KAREL ZEMÁNEK

VEDOUCÍ PRÁCE
SUPERVISOR
Mgr. PAVEL RAJMIC, Ph.D.

BRNO 2008

Abstrakt

Tato bakalářská práce se zabývá realizací waveletové transformace v reálném čase v prostředí Pure Data. První kapitola je věnována teoretickému minimu waveletové transformace. Druhá kapitola je věnována programu Pure Data. Jsou zde uvedeny jednoduché patche, ale i postup jak psát vlastní externí programy použitelné programem Pure data. Třetí kapitola se zabývá realizací waveletové transformace v programu Pure Data. Obsahuje vybraný algoritmus pro výpočet waveletové transformace i popis vlastního programu.

Klíčová slova

Pure Data, waveletová transformace, zpracování signálů

Abstract

The bachelors thesis considers of wavelet transform realization in the real-time in Pure Data. The first part of the thesis is devoted to the theoretical minimum of wavelet transform. The second part is devoted to program Pure Data itself. You can see simply patches and also instruction for external programs compact to PD. The third part is devoted to realization of wavelet transtorm in Pure Data program. It contain specific algorithm for wavelet transform calculation and description of exist program.

Key words

Pure Data, wavelet transform, signal processing

LICENČNÍ SMLOUVA

POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Karel Zemánek

Bytem: sídl. Družba 1209, Brumov-Bylnice

Narozen/a (datum a místo): 28.1. 1986 ve Zlíně

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - □ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako

   .....

   (dále jen VŠKP nebo dílo)

Název VŠKP:	Waveletová transformace v reálném čase v prostředí Pure Data
Vedoucí/ školitel VŠKP:	Mgr. Pavel Rajmic, Ph.D.
Ústav:	Telekomunikací
Datum obhajoby VŠKP:	

VŠKP odevzdal autor nabyvateli v*:

- □ tištěné formě – počet exemplářů 1
- □ elektronické formě – počet exemplářů 1

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - □ ihned po uzavření této smlouvy
   - □ 1 rok po uzavření této smlouvy
   - □ 3 roky po uzavření této smlouvy
   - □ 5 let po uzavření této smlouvy
   - □ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: ……

……. …….

Nabyvatel Autor

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma „Waveletová transformace v reálném čase v prostředí Pure Data“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ..... ……..

podpis autora

Děkuji vedoucímu bakalářské práce Mgr. Pavlu Rajmicovi, Ph.D., za velmi užitečnou metodickou pomoc a cenné rady při zpracování bakalářské práce.

V Brně dne …… ….....

SEZNAM POUŽITÝCH ZKRATEK, VELIČIN A SYMBOLŮ

CWT	integrální waveletová transformace (Continuous Wavelet Transform)
DP	filtr typu dolní propust
D/A	digitálně – analogový
DP rek.	rekurzivní filtr typu dolní propust
DWT	diskrétní waveletová transformace (Discrete Wavelet Transform)
DTWT	konečná diskrétní waveletová transformace (Discrete Time Wavelet Transform)
floor	zaokrouhlení na nejbližší celé číslo směrem k nule
HP	filtr typu horní propust
HP rek.	rekurzivní filtr typu horní propust
IDTWT	zpětná konečná diskrétní waveletová transformace (Inverse Discrete Time Wavelet Transform)
MR-analýza	mnohaměřítková analýza (Multiresolution Analysis)
$L^2(R)$	množina všech komplexních funkcí lebesgueovsky integrovaných v kvadrátu s konečnou energií
PD	Pure Data
SegWT	segmentovaná waveletová transformace (Segmented Wavelet Transform)
Z	množina celých čísel
ψ	waveletová funkce
ϕ	měřítková funkce
$\oplus$	operátor ortogonálního součtu vektorových prostorů

SEZNAM OBRÁZKŮ

OBSAH

ÚVOD

V prvním kapitole mé bakalářské práce se budu zabývat problematikou waveletové (nebo-li vlnkové) transformace, která má velkou možnost uplatnění, zejména se používá na separaci signálu od šumu. Oproti Fourierově analýze má velkou výhodu. Při Fourierově analýze transformujeme do kmitočtové roviny a přitom ztrácíme zcela informace o časové rovině. Waveletová transformace nám nabízí nástroj, kterým můžeme signál sledovat jak v kmitočtové rovině tak i v časové. Jedná se o mladou metodu v oblasti zpracování signálů a i přesto se s ní dnes setkáváme velice často, například při kompresi digitálních dat, jako je obraz a video. V dnešní době se ale waveletová transformace používá převážně jen v aplikacích běžících „off-line“, kde předem známe délku celého signálu.

V další kapitole své práce se budu věnovat programu Pure Data, vytvořeného Millerem Puckettem. Dokumentaci i zdroje na stažení programu naleznu na internetových stránkách Millera Pucketta. Jedná se o real-time program na zpracování, úpravu a tvorbu hudby. Ale jelikož je to open-source program s dobrým interface, jsou jeho možnosti využití takřka neomezené. Existuje spousta nadšenců i profesionálů po celém světě, pracujícími s tímto programem. Proto se program každým rokem zdokonaluje a je mu přidávána jedna funkce za druhou právě samostatnými uživateli.

Cílem mé bakalářské práce je realizace waveletové transformace v prostředí Pure Data. Jak již bylo nastíněno, program Pure Data je real-time program, tudíž budu ve své práci zpracovávat waveletovou transformaci v reálném čase. Budu muset zvolit vhodný algoritmus výpočtu waveletové transformace, který nebude tolik náročný na čas a bude dostatečně přesný. Dále ve své práci budu muset zvážit, zda bude možné realizovat waveletovou transformaci prvky obsaženými v programu Pure Data, nebo zda si doprogramovat část vlastního kódu použitelného v Pure Data. Obě možnosti vedou k důkladnému prostudování vlastností a možností programu. Všechny tyto znalosti, které povedou k realizaci, uvedu v mé bakalářské práci.

1 Waveletová transformace

Základní rozdíl mezi Fourierovou a waveletovou transformací (WT) je v tom, že wavelety nejsou periodické funkce. Posunutím a dilatací mateřského waveletu lze funkci lokalizovat jak ve frekvenční tak i v časové oblasti. Tato vlastnost dělá WT velmi výhodnou pro analýzu nestacionárních nebo aperiodických jevů. WT se používá v různých oborech jako je například komprese obrazu (JPEG 2000), analýza řeči, rekonstrukce poškozených nahrávek nebo odhalování poruch.

1.1 Integrální waveletová transformace

V integrální waveletové transformaci (CWT) je funkce ψ, která prakticky vypadá jako malá vlnka, použita na vytvoření rodiny waveletů $\psi(at+b)$. Kde *a* a *b* jsou reálná čísla. Číslo *a* rozšiřuje (roztahuje nebo smršťuje) fucnkci ψ a číslo *b* ji posouvá.

CWT mění signál f (t) na funkci c(a,b) dvou proměnných, měřítka a času.

$$c(a,b) = \int_{-\infty}^{\infty} f(t) \cdot \psi_{a,b}(t)\, dt \tag{1.1}$$

Kde $\psi_{a,b}(t)$ nazýváme bázovou funkcí. Dle literatury [6] můžeme bázovou funkci zapsat ve tvaru:

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \cdot \psi\left(\frac{t-b}{a}\right) \tag{1.2}$$

a udává měřítko, *b* posunutí. A vztah $\frac{1}{\sqrt{a}}$ je zde kvůli zachování normy v $L^2(R)$

(tj. energie) tak, aby pro každé přípustné *a* platil vztah (1.2) .

Integrální waveletová transformace (se spojitým časem) je tedy dle [6] definována vztahem:

$$c(a,b) = \frac{1}{\sqrt{a}} \cdot \int_{-\infty}^{\infty} f(t) \cdot \psi\left(\frac{t-b}{a}\right) dt \tag{1.3}$$

Mateřský wavelet ψ musí mít nulovou střední hodnotu

$$\int_{-\infty}^{\infty} \psi(t)\, dt = 0 \tag{1.4}$$

Jak vyplývá ze vzorce (1.4), wavelety musí mít oscilační charakter a jejich kmity musí být tlumeny směrem ke kladnému i zápornému nekonečnu.

1.2 Diskrétní waveletová transformace

U diskrétní waveletové transformace (DWT) je wavelet rozšířen nebo roztáhnut pouze o diskrétní hodnoty. Nejčastěji jsou rozšířeny o mocninu dvou, potom je nazýváme též dyadické dilatace. Funkce $\psi_{j,k}$ je pak pro každá celá čísla j a k definována předpisem

$$\psi_{j,k}(t) = 2^{-\frac{j}{2}} \psi\left(2^{-j} t - k\right) \tag{1.5}$$

Potom dle litaretury [4] definujeme DWT vzorcem (1.6). Můžeme též říci, že DWT je vzorkovanou verzí CWT s parametry a=2^j, b=$2^j k$, $j,k \in Z$.

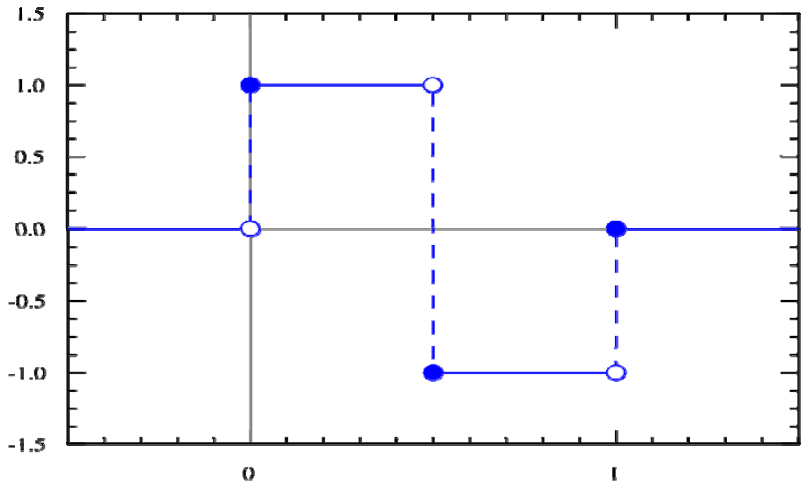
$$DWT_f(j,k) = \left\langle f, \psi(j,k) \right\rangle = \int_{-\infty}^{\infty} f(t).\psi_{j,k}(t)\,dt \tag{1.6}$$

Wavelet je většinou konstruován tak, aby byl systém $\{\psi(j,k)\}_{j,k\in Z}$ ortonormální v prostoru $L^2(R)$, ale jsou i jiné možnosti. Ortonormalita systému znamená, že informace obsažena ve waveletovém koeficientu není obsažena v žádném jiném waveletovém koeficientu. Původní signál také můžeme plně zrekonstruovat pomocí waveletových koeficientů. Pokud je podmínka ortonormality splněna, můžeme libovolnou funkci $f(t) \in L^2(R)$ vyjádřit dle [4] součtem waveletové řady

$$f(t) = \sum_{j,k \in Z} \left\langle f, \psi_{j,k} \right\rangle \psi_{j,k}(t) \tag{1.7}$$

Číslo $c_{j,k} = \left\langle f, \psi_{j,k} \right\rangle$ je pojmenováno jako j,k-tý waveletový koeficient. Množinu $\{c_{j,k}\}_{j,k\in Z}$ nazýváme waveletové spektrum funkce f a funkci ψ říkáme mateřský wavelet.

Byly nalezeny ortonormální báze $L^2(R)$ takové, že její prvky mají kompaktní nosič, tj. lze nalézt ohraničený interval, že mimo něj má funkce nulovou hodnotu. Příkladem takovéto funkce je tzv. Haarův wavelet (obr 1.2) nebo wavelet typu Daubechies.



Obr. 1.2 Haarův wavelet

1.3 MR – analýza

Pojem MR – analýza pochází z anglického názvosloví multiresolution analysis. Je to mnohaměřítková analýza pro rozklad signálu do bázových funkcí různých měřítek.

Pro přípustný wavelet ψ lze tento rozklad do bázových prostorů dle [6] zapsat

$$L^2(R) = ... \oplus W_2 \oplus W_1 \oplus W_0 \oplus W_{-1} \oplus ..... \tag{1.8}$$

kde $\oplus$ značí ortogonální součet prostorů funkcí. Existuje-li navíc měřítková funkce, která generuje MR – analýzu, potom platí vztah

$$V_j = V_{j+1} \oplus W_{j+1} \tag{1.9}$$

kde prostor W_{j+1} je ortogonální na prostor V_{j+1} a reprezentuje diferenci mezi V_1 a V_{j+1}.

Potom lze vzorec 1.8 upravit pro libovolné zvolené j na

$$L^2(R) = V_0 \oplus W_j \oplus W_{j-1} \oplus W_{j-2} \oplus ..... \tag{1.10}$$

Základem MR – analýzy jsou dva důležité vztahy, rovnice měřítkové funkce

$$\phi(t) = \sqrt{2} \sum_{k=-\infty}^{\infty} h_k \phi(2t - k) \tag{1.11}$$

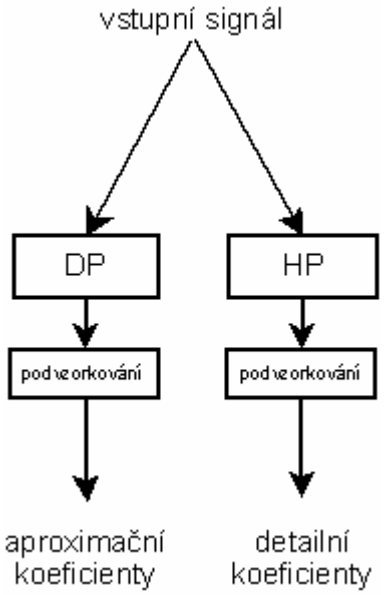
a waveletová rovnice

$$\psi(t) = \sqrt{2} \sum_{k=-\infty}^{\infty} g_k \phi(2t - k) \tag{1.12}$$

Pro přesné vysvětlení a matematické odvození viz [4, kap. 4]

1.4 Mallatův pyramidový algoritmus

V některé literatuře se setkáme též z názvem rychlá waveletová transformace, podle analogie s rychlou fourierovou transformací. Vstupní signál pustíme do dvojice číslicových filtrů. Jeden filtr je typu dolní propust h, druhý typu horní propust g.Tyto filtry nazýváme kvadraturní zrcadlové filtry. Dolní propust h v sobě skrývá koeficienty měřítkové funkce a horní propust g koeficienty příslušného waveletu. Výstupy z těchto filtrů decimujeme, tj. vezmeme pouze jeden vzorek ze dvou. Získané koeficienty z dolní propusti nazýváme aproximačními waveletovými koeficienty a koeficienty z horní propusti detailními waveletovými koeficienty. Tím jsme provedli jeden krok waveletové dekompozice. Znázornění tohoto kroku je na obrázku 1.4.1 .

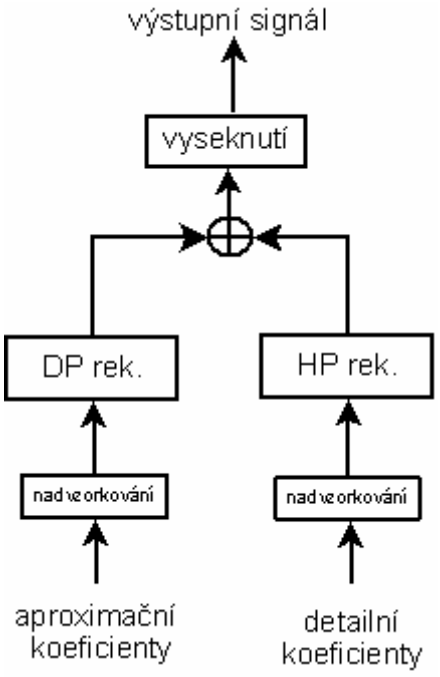


Obr 1.4.1 Schéma jednoho stupně waveletové kompozice.

Detailní koeficienty uchováme a aproximační koeficienty můžeme stejným způsobem podrobit filtraci. Počet provedených filtrací nazýváme hloubkou dekompozice. Pokud má vstupní signál délku s, potom je maximální hloubka dekompozice $d_{\max}$ omezena vztahem

$$d_{\max} \leq \log_2 s \tag{1.13}$$

Algoritmus pro rekonstrukci je podobný algoritmu pro dekompozici. Opět použijeme filtry. Ale než k nim přivedeme signál, musíme jej nadvzorkovat. To se děje tak, že za každý vzorek vložíme další s nulovou hodnotou. Takto interpolovaný signál filtrujeme pomocí rekurzivních filtrů. Tyto filtry značíme jako g' a h' .Výsledek z těchto dvou filtrů sečteme a vysekneme prostřední část. Tím získáme aproximační koeficienty úrovně o jedna vyšší než vstupní. Zpětný algoritmus rekurze provádíme tolikrát, kolik je hloubka dekompozice. Jeden stupeň dekompozice je uveden na obrázku 1.4.2 .



Obr. 1.4.2 Jeden krok waveletové rekonstrukce.

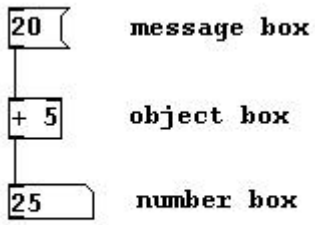
2 Pure Data

Pure Data (PD) je prostředí pro zpracování hudebního signálu a prioritně pro tvorbu vlastní hudby. Může být použit rovněž pro jakákoliv jiná média. PD je open-source program, vytvořený Millerem Puckettem. V posledních verzích programu jsou již doplňky doprogramovány samotnými uživateli.

PD dokumenty se nazývají „patches“ . Vypadají jako různě pospojované obdélníkové schránky. Při bližším pohledu na tyto zdánlivě stejné schránky zjistíme, že vypadají každá trošku jinak a při ještě detailnějším pohledu zjistíme, že i stejně vypadající schránky mohou mít jiné vlastnosti. PD je totiž realizační nástroj, nikoliv specifický programovací jazyk. PD patch se skládá ze sbírky schránek propojených v jakousi síť. Lem schránky nám říká, jak je text ve schránce překládán a jak schránka funguje. Zde uvedené informace jsem čerpal z [3].

2.1 Základní funkce schránek v PD

Na jednoduchém příkladě na obrázku 2.1 vysvětlím základní tři typy schránek. Kvůli zažitým termínům jsem ponechal názvy schránek v anglickém jazyce.



Obr. 2.1 Příklad tří různých schránek v PD.

Z obrázku už je jasně patrný rozdíl mezi jednotlivými schránkami. A to jak po stránce vzhledu jednotlivých schránek, tak i jejich funkčnosti.

Popis jednotlivých částí :

Message box

Tato schránka překládá text, který je do ní předem vložený, a posílá ho jako zprávu pokaždé, když je tato aktivována kliknutím na ni v editačním módu. V našem případě se odešle číslo 20 všude tam, kam je tato schránka připojena.

Object box

Tato schránka vyhodnotí text uvnitř a při načítání vytvoří daný objekt. Objektové schránky můžou nabývat stovky různých objektových tříd. Například z nich můžeme vytvořit oscilátory, násobitele, D/A měniče a spoustu dalších. Tato schránka je vlastně jakýmsi stavebním kamenem, z něhož si můžeme vytvořit libovolnou stavbu. První slovo nebo znak vepsaný do schránky označuje její třídu. Zde na tomto příkladu je jako první znak znaménko „+“ , které označuje, že se jedná o sčítací člen. Další znaky musí být odděleny stiskem klávesy mezerník. Říká se jim argumenty vytvoření. Tyto argumenty specifikují počáteční stav při vytváření objektu. V našem jednoduchém příkladě je to číslice 5, která se vždy přičte k příchozím datům.

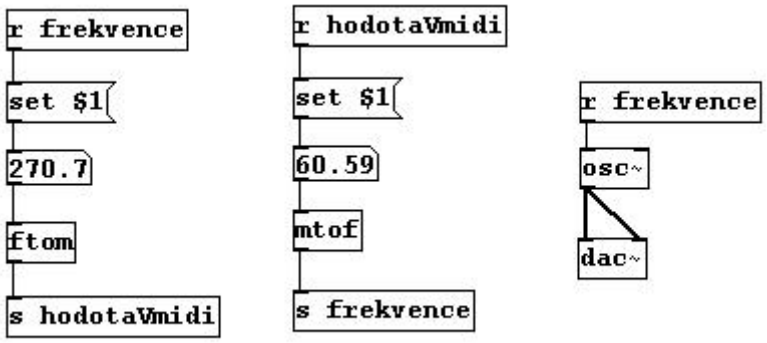
Number Box

Je zvláštním typem GUI schránky. Mimo jiné se dá použít jak tlačítko nebo pákový spínač. Zatímco je hodnota v object a message boxu stále pevně držena, u numer boxu se mění a odráží tak aktuální hodnotu přiváděnou do schránky. Tato schránka se dá samozřejmě použít i pro odesílání dat. Její obsah ale před spuštěním (přechodem do editačního módu) bude nula. Obsah se mění až za běhu programu, a to buď přímým zadáním z klávesnice nebo kliknutím a následným tahem myši. To nám umožní za běhu programu libovolně měnit hodnoty, které chceme poslat na další zpracování.

Funkce tohoto ilustračního příkladu je tedy jasná. Před přepnutím do editačního módu zadám hodnotu do message boxu a taky nastavím hodnotu v object boxu. Po přepnutí do editačního módu je v number boxu stále hodnota nula. Po kliknutí myší na message box se odešle hodnota 20 do object boxu kde se sečte s hodnotou 5. Object box poté hned odešle sečtenou hodnotu do number boxu. Ten se okamžitě nastaví na hodnotu 25.

2.2 Jednoduchý příklad použití PD

Na dalším příkladu předvedu jak se dá pracovat s PD a jak se zde dají posílat zprávy aniž by se musely jednotlivé schránky propojovat. Pokusím se to vysvětlit na obrázku 2.2.



Obr. 2.2 Posílání zpráv bez propojení schránek a příklad převodu MIDI na frekvenci.

Hlavní roli zde hrají object boxy s počátečnímy písmeny *s* a *r* . Podle anglických slov send a recive. První schránka, tak zvaný recive box (na obrázku vlevo nahoře), čeká na nelokální zprávu od send boxu . V tomhle případě čeká na zprávu, kterou jsem si pojmenoval frekvence. Schránka začínající na *s* posílá nelokální zprávy. Příjemce, který začíná klíčovým znakem *r*, je vyhodnotí podle ukazatele. Na obrázku 2.2 máme dvě vysílací schránky s ukazateli frekvence a hodotaVmidi. Recive box se stejným ukazatelem zprávu převezme a odešle dál podle toho, kam je připojen.

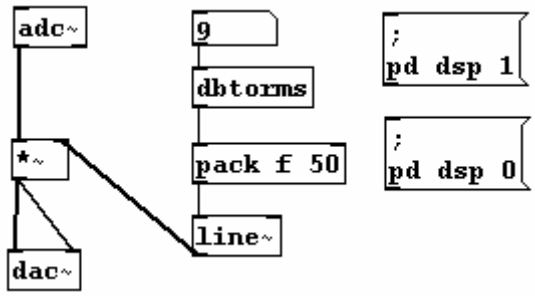
Celá funkce tohoto demonstrativního příkladu je převod mezi frekvencí a MIDI číslem tónu a následná interpretace tónu pomocí oscilátoru. Levý sloupec přijme zprávu o frekvenci a zobrazí ji v number boxu. Mezi recive boxem a number boxem je vložen message box s hodnotou $1. Přednastavená hodnota $1 slouží k tomu, aby nám nevznikla nekonečná nestabilní smyčka mezi levým a pravým sloupcem. Pokaždé když přijde zpráva z recive boxu přepíše message box svou aktuální hodnotu a odešle ji do number boxu, kde se nám zobrazí hodnota frekvence v hertzích. Tuto hodnotu ale můžeme během běhu programu tahem myší změnit. Tato hodnota se okamžitě převádí pomocí object boxu, který má uvnitř vepsané slovo ftom. Tato schránka převádí aktuálně přiváděnou hodnotu frekvence na hodnotu MIDI. Object box mtof dělá přesný

opak, tudíž převádí MIDI na frekvenci. Tyto převodníky jsou velice náročné na procesorový čas. Proto jejich použití výrazně zpomaluje aplikace, což je u programů pracujících v reálném čase dosti nežádoucím jevem.

Jako kontrolu námi zvolených převodů jsem použil oscilátor, který generuje tón požadované frekvence. Je řízený právě příchozí hodnotou frekvence z recive boxu. Jedná se opět o object box s klíčovým slovem osc~ . Tento oscilátor pracuje jak na vstupu tak na výstupu s digitálním signálem. Generuje sinusový průběh, a pokud není na jeho levou stranu přivedena jiná hodnota, má výstup amplitudu jedna. Hodnota frekvence se dá nastavit buď pevně, napsáním hodnoty frekvence v hertzích za klíčové slovo osc~, nebo přivedením hodnoty na levou část vstupu. Tak je to provedeno i na obrázku 2.2 . Poslední object box má klíčové slovo dac~ . Jedná se o zvukové výstupní zařízení. Záleží na našem hardwaru. Vůbec se nemusí jednat o digitální/analogový převodník jak by mohl název napovídat, ale obecně nám dovolí posílat nějaký zvukový signál na zvukový výstup našeho počítače.

2.3 Příklad vstupu a výstupu zvukového signálu

Jako poslední jsem uvedl jednoduchý příklad, který je uveden na obrázku 2.3. Jedná se o jednoduché řízení hlasitosti vstupního signálu braného z mikrofonu, který se posílá na výstup, tj. do reproduktorů, případně do sluchátek.



Obr. 2.3 Ovládání výstupní hlasitosti vstupního signálu.

Celý program je jednoduchý. Obsahuje pouze pár základních schránek. Jako první v levém sloupci je object box s klíčovým slovem adc~ . Opět se nejedná o analogovo/číslicový převodník, jak by název mohl napovídat. Tenhle object box slouží

jako vstup signálu z vnějšího zařízení připojeného k naší zvukové kartě. Tudíž opět záleží na našem hardwaru. Dále je tu object box s klíčovým znakem *~ , který je označován jako násobitel. Jestliže je na pravou stranu schránky přiváděn nějaký argument (tak jako v našem případě), tak tato schránka násobí digitální zvukový signál, přiváděný do levého vstupu, s číslem přiváděným na pravý vstup. Pokud nepřivádíme číslo ale signál, násobí tato schránka dva přiváděné signály mezi sebou.

Prostřední sloupec má jednoduchou funkci. A tou je posílat do násobitele požadovanou hodnotu hlasitosti výstupního signálu v decibelech. Pro zadávání hodnoty a také pro změnu hodnoty během běhu programu nám slouží horní number box. V něm už uvádíme požadovanou hodnotu v decibelech. Jako další schránka je object box s klíčovým slovem dbtorms. Je to převod hodnoty v decibelech na velikost amplitudy. Proto znak jako je rms by spíše měl obsahovat klíčové slovo amp, aby nedocházelo ke zbytečnému matení uživatelů. Jako další schránka je object box s klíčovým slovem pack. Funkce téhle schránky spočívá v tom, že na svůj výstup pošle dvojici čísel. V našem případě výstup ze schránky dbtorms a 50. Tuhle zprávu převezme schránka s klíčovým slovem line~, která plynule naroste na hodnotu, která je uvedena v prvním parametru za dobu 50 milisekund, což je hodnota druhého parametru.

Do tohoto programu sem též zařadil dvě schránky, které zdánlivě k programu vůbec nepatří. Jsou to dva message boxy, které se dají použít jako zapínaní a vypínaní audio signálu přiváděného a odváděného do výstupního zařízení (v našem případě do zvukové karty počítače). Tento úkon se vždy musí udělat v hlavním menu PD označeném média a tam teprve zapnout nebo vypnout audio. Proto je pro uživatele daleko příjemnější (i mnohem rychlejší) kliknout myší na ikonu přímo v patchy než přepínat mezi okny k hlavnímu oknu PD a tam ještě hledat v záložce.

Ještě je hodno podotknout, že signál je digitální, tudíž navzorkovaný. Vzorky se posílají po segmentech. Pokud není nastaveno jinak, je v základním nastavení segment o délce 64 vzorků. Pomocí object boxu s klíčovým slovem block~ [n] , kde místo n uvedeme kolik vzorků má segment obsahovat. Tato informace je dosti důležitá, neboť při realizaci waveletové transformace v realném čase budu zpracovávat signál po segmentech.

2.4 Jak psát externí programy pro PD

Pro psaní vlastních vnějších programů (externals) je nezbytný kompilátor jazyka C, který podporuje ANSI-C standart, jako například Gnu C-Compiler pro linuxové systémy, nebo Visual C++ pro operační systém Windows.

PD je napsán v programovacím jazyce C. Vzhledem k jeho grafické povaze je PD objektově orientovaný systém. Bohužel programovací jazyk C moc dobře nepodporuje používání tříd, proto nebude výsledný zdrojový kód nikdy tak elegantní.

Základní pojmy pro vytváření programů v PD jsou definovány v [3].

Internal – internal je třída vytvořená přímo v PD. Jedná se o primitivní funkce jako je například „+“ .

External – external je třída která není vytvořená přímo v PD, ale je nahrána při běhu programu. Když je nahrán do paměti PD, nedá se již od internalu rozlišit.

Knihovna – knihovna je sbírka externals, která je zkompilovaná do jednoho binárního souboru.

Na rozdíl od externals se musí knihovny importovat do PD speciálními operacemi. Poté co je knihovna importována se všechny v ní obsažené externals nahrají do paměti PD a jsou dostupné jako objekty.
Knihovny se dají do PD vložit dvěma způsoby :

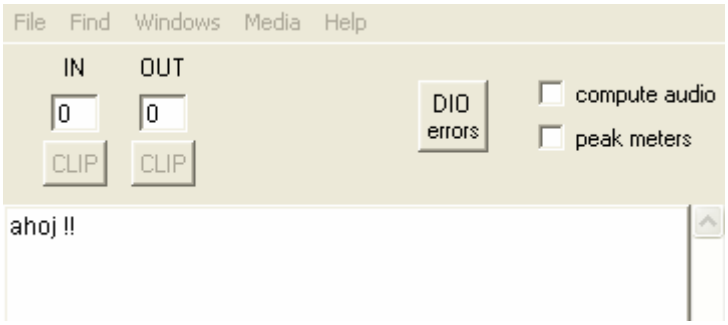
- Přímým zapsáním do příkazového řádku: –lib moje_knihovna
- Vytvořením objektu moje_knihovna

První způsob nahraje knihovnu při startu PD. Tohle je většinou používáno pro knihovny které obsahují více externals. Druhá metoda je vhodná pro knihovny, které obsahují pouze jeden external se shodným jménem jako knihovna. PD první zkontroluje, zda-li je třída se jménem moje_knihovna už nahrána. Pokud ne, začne hledat soubor s názvem moje_knihovna.dll. Když nalezne hledaný soubor, jsou všechny v něm obsažené

externals nahrány do paměti zavoláním funkce moje_knihovna_setup(). Pokud PD nenalezne požadovanou knihovnu nebo se nepodaří nahrát external do paměti, vypíše se chybová hláška do základního okna PD.

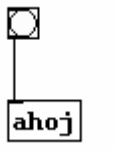
2.5 Jednoduchý externí program pro PD

Funkce toho programu je naprosto jednoduchá. Po kliknutí na bang-schánku se v základním okně PD vypíše hláška *ahoj !!*. Základní okno s toouto hláškou je vidět na obrázku 2.5. 1



Obr. 2.5 .1 Základní okno PD s hláškou ahoj !!

Patch v PD obsahuje pouze dvě schránky, a to bang-schánku a object box, jenž bude obsahovat vytvořený external.



Obr. 2.5.2 Patch s vytvořeným externalem.

Vytvoření externího programu ahoj v jazyce C

K vytvoření vlastního programu je potřeba mít dobře definované rozhraní. O to se stará soubor m_pd.h, který se musí při psaní programu přidat mezi hlavičkové soubory. Jako první si musíme připravit datový prostor a nadefinovat novou třídu.

```
static t_class *ahoj_class;
typedef struct _ahoj{
t_object x_obj;
} t_ahoj;
```

Ahoj_class je ukazatel na novou třídu, struktura t_ahoj je datový prostor pro tuto třídu. Nezbytným prvkem datového prostoru je proměnná typu t_object, která je používána k uložení interních vlastností objektu, jako je například grafická prezentace objektu nebo data kolem inletů a outletů (vstupů a výstupů object boxu). Proměnná typu t_object by měla být prvním vstupem do struktury.

Kromě datového prostoru potřebuje třída sadu manipulátorů (metod) k manipulaci s daty. Metoda se zavolá tehdy, pokud je do instance naší třídy poslána zpráva. Metody jsou jakýmsi rozhraním do systému zpráv v PD a už podle principu je jasné, že nebudou mít žádnou návratovou hodnotu.

```
void ahoj_bang(t_ahoj *x){
post("ahoj !!");
}
```

Příkaz post( ) zde má stejnou funkci jako například printf( ) v jazyku C s tím rozdílem, že post vypíše znakový obsah v závorce do chybového řádku hlavního okna PD.

K vytvoření nové třídy potřebujeme PD předat informace o datovém prostoru a metodách. To se děje když PD nahrává naši knihovnu. Když se nahrává, PD se snaží zavolat funkci moje_knihovna_setup( ). Tato funkce se opět volá jen jednou a to právě při načítání knihovny. Pokud volání téhle funkce selže, nebude se již volat podruhé a žádný external z naší knihovny se nenačte.

```
void ahoj_setup(void)
{
  ahoj_class = class_new(gensym("ahoj"),
     (t_newmethod)ahoj_new,
     0, sizeof(t_ahoj),
     CLASS_DEFAULT, 0);
 class_addbang(ahoj_class, ahoj_bang);
}
```

Funkce class_new vytvoří novou třídu a vrátí ukazatel na tento prototyp. První argument této funkce je symbolické jméno třídy, tedy jméno které budeme v PD vepisovat do object boxu, v tomto případě se do object boxu vepíše ahoj. Další dva argumenty definují konstruktor a dekonstruktor. Pokaždé, když je objekt vytvořený v patchy PD, konstruktor (t_newmethod)ahoj_new inicializuje prostor pro data. Kdykoliv je objekt zničen, zavřením příslušného patche nebo vymazáním daného object boxu, dekostruktor uvolní dynamicky přidělenou paměť. Zároveň je uvolněna i alokovaná paměť rezervovaná pro statický datový prostor. Protože v našem příkladě nepotřebujeme pracovat s datovým prostorem, není třeba destruktor a argument je nastaven na 0. Abychom umožnili PD reservovat a uvolňovat dostatek paměti pro statický datový prostor, je třeba, abychom jako čtvrtý argument uvedli velikost datové struktury. Pátý argument má vliv na grafickou representaci objektu, zde jsem ponechal základní hodnotu CLASS_DEFAULT. Poslední argument definuje argumenty objektu a jejich typy. Může nabývat až šesti numerických nebo symbolických hodnot. Více o jednotlivých symbolech je popsáno v [3]. Na posledním řádku přidáváme příkazem class_addbang metodu pro zacházení s bang zprávami třídě, která je uvedena jako první argument.

Pokaždé, když je v patchy vytvořený objekt, konstruktor definovaný v příkazu class_new vytvoří novou instanci třídy. Konstruktor musí být typu void *.

```
void *ahoj_new(void)
 { t_ahoj *x=(t_ahoj *)pd_new(ahoj_class);
   return (void *)x;
 }
```

Tento jednoduchý program sloužil jen pro základní vysvětlení jak se vytvářejí externí programy pro PD. Nepracuje se v něm se žádnými speciálními třídami pro práci se signálem.

Při realizaci waveletové transformace však bude potřeba pracovat se signálovými třídami a také bude třeba vytvořit prostředí pro inlety a outlety. Při vytváření signálových objektů (signálový objekt poznáme tak, že má za klíčovým slovem ~) je třeba nadefinovat tzv. DSP metodu. Když se v PD zapne audio (v liště audio se stiskne audio on), všechny objekty které obsahují dsp metodu jsou identifikovány jako signálové.

2.6 Metody pro vytváření externích programů pro PD pracujících se signály

Dsp metoda

Pokud je v PD spuštěno audio, všechny signálové objekty deklarují svoje perform-routine aby mohly být přidány do DSP stromu celého PD. DSP metoda má dva argumenty, ukazatel na datový prostor třídy a ukazatel na pole signálu.

```
void moje_dsp_method(t_mojedata *x, t_signal **sp)
  {
    dsp_add(mojetrida_perform, 5, x,
                sp[0]->s_vec, sp[2]->s_vec, sp[0]->s_n);
  }
```

Dsp_add přidá třídu mojetrida do DSP-stromu PD. Druhý argument je počet následujících ukazatelů na různé proměnné. Počet těchto ukazatelů není omezen. Stejně jako není dáno, který se má přiřadit ke které proměnné. Dále jsou zde deklarovány jeden vstupní a jeden výstupní signál. Pokud máme více vstupních a výstupních signálu definují se takhle :

- sp[0] signál vstupující do levého inletu

- sp[1] signál vstupující do pravého inletu
- sp[2] signál vytupující z pravého outletu
- sp[3] signál vytupující z levého outletu

Jsou buď typu s_n, což je délka signálového vektoru, nebo s_vec, což je ukazatel na signálový vektor. Pokud pracujeme v rámci jen jedno patche bez žádných subpatchů, stačí nadefinovat délku jen pro jeden vstupní vektor (jelikož všechny vektory v jednom patchy mají shodnou velikost, v základu nastavenou na 64 vzorků). Takže struktura typu t_signal se skládá z ukazatele na signálový vektor .s_vec (pole vzorků typu t_sample) a délkou tohoto vektoru .s_n.

3 Waveletová transformace v reálné čase v prostředí Pure Data.

Po prostudování možností PD a po poradě s vedoucím bakalářské práce jsem došel k závěru, že realizace waveletové transformace pomocí objektů, které PD obsahuje, by byla velice složitá a nepřehledná. Teoreticky by se uskutečnit dala, ovšem PD neumožňuje provést kontrolu chyb a následné ladění programu nebo hledání závad by bylo skoro nemožné. Řešení tudíž spočívá ve vytvoření vlastního externího programu. Z tohoto programu se vhodným zkompilováním vytvoří dynamická knihovna s příponou .dll a zkopíruje se do adresáře kde je nainstalováno PD. Postup jak tyto externí programy psát je uveden v předchozí kapitole.

3.1 Výběr vhodného algoritmu

Pro výpočet waveletové transformace jsem zvolil algoritmus DTWT. Pro realizaci v reálném čase to není úplně ideální, neboť nevíme jak dlouhý bude zpracovávaný signál. Signál který budu zpracovávat v PD je, jak již jsem zmínil na konci kapitoly 2.3, navzorkován a posílán po segmentech. Ideálním řešení by bylo použití segmentované waveletové transformace (SegWT). Jedná se o novou metodu zpracování waveletové transformace v reálném čase .Pro více informací o SegWT viz [4]. Algoritmus SegWT je však poměrně složitý a po problémech spojených s nemožností aplikovat waveletovu transformaci přímo prostředky PD a následných problémech s vytvářením dynamických knihoven pro PD jsem po radě svého vedoucího využil algoritmus DTWT.

3.1.1 Algoritmus DTWT

Zde uvedený algoritmus jsem převzal z [4, kap.8]

Je dán vstupní signál x délky c, dva waveletové dekompoziční filtry délky m, horní propust g a dolní propust h, je zvolena hloubka dekompozice d a rovněž je dán tzv. typ prodloužení signálu.

- označíme vstupní signál jako $a^{(0)}$. Nastavíme j = 0.

- jeden krok dekompozice :
  1. Rozšíření vstupního vektoru. Prodloužíme vektor $a^{(j)}$ nalevo i napravo o (m-1) vzorků, podle zvoleného typu prodloužení.
  2. Filtrace. Filtrujeme prodloužený signál fitrem g, což můžeme vyjádřit jejich konvolucí.
  3. Vyseknutí. Z výsledku filtrace vybereme jen její prostřední část tak, že zbytky vlevo i vpravo jsou shodné délky (m-1) vzorků.
  4. Podvzorkování (decimace). Vyseknutý vektor decimujeme, tj. ponecháme pouze jeho vzorky se sudými indexy.

  Uložíme tento vektor jako $d^{(j+1)}$. Opakujeme body 2. – 4., nyní s filtrem h a výsledek uložíme jako $a^{(j+1)}$
- zvýšíme j o jedničku. Pokud nyní j < d jdeme na bod 2.,jinak algoritmus končí.

Poznámky k algoritmu:

- po skončení algoritmu máme uloženy waveletové koeficienty ve vektorech $a^{(d)}, d^{(d)}, d^{(d-1)}, d^{(d-2)}$, ….., $d^{(1)}$
- ad 1. prodloužení je nutno vykonat kvůli jednoznačnosti transformace, a to v každém kroku dekompozice
- ad 1. ve speciálním případě tzv. periodického prodloužení je délka potřebného prodloužení signálu menší než (m-1). Periodické prodloužení však pro naše účely není relevantní a proto jej dále neuvažujeme
- ad 1. v prvním kroku algoritmu tedy prodloužíme $a^{(0)}$ na délku $s + 2(m-1)$
- ad 2. délka výsledné konvoluce v prvním kroku je $\left[s + 2(m-1) + m - 1\right] = s + 3(m-1)$
- ad 3. prostřední část má tedy délku, jakou by měla konvoluce neprodlouženého vektoru $a^{(j)}$ s waveletovým filtrem. V prvním kroku je vyseknutá část délky $s + m - 1$
- ad 4. touto operací dostáváme v prvním kroku vektor o délce $floor\left[(s + m - 1)/2\right]$

Aby byla DTWT použitelná, je zapotřebí z koeficientů znovu na výstupu „poskládat“ signál. K tomuto potřebujeme zpětnou diskrétní waveletovou transformaci (IDTWT). Pokud nebudeme měnit žádný z koeficientů, zpětně zkonstruovaný signál se od původního signálu nebude lišit. Algoritmus musí být opět velice rychlý aby stíhal zpracovávat data přiváděná od DTWT a také aby nevnášel do zpracovávaného signálu velké zpoždění.

3.1.2 Algoritmus IDTWT

Zde uvedený algoritmus jsem převzal z [4, kap. 8]

Jsou dány vstupní koeficienty, aproximace $a^{(n)}$ a detaily $d^{(n)}$, výstupní signál v, dva waveletové rekonstrukční filtry délky m, horní propust g' a dolní propust h', je známa hloubka dekompozice a rovněž je dán typ prodloužení signálu.

- Označíme vstupní koeficienty $a^{(j)}$ a $d^{(j)}$. Nastavíme j = n.
- Jeden krok rekonstrukce:

  1. Nadvzorkování. Vektor $a^{(j)}$ proložíme nulami na sudé pozice, tj. vstupní signál budou tvořit vzorky s lichými indexy.
  2. Filtrace. Filtrujeme prodloužený vektor $a^{(j)}$ filtrem g', což můžeme vyjádřit jejich konvolucí.

  Opakujeme body 1. – 2. pro vektor $d^{(j)}$ nyní však s filtrem h'

  3. Sečtení vstupních vektorů. Sečteme vstupní vektory $a^{(j)}$ a $d^{(j)}$ vzorek po vzorku.
  4. Vyseknutí. Z výsledku sečtení vybereme jen její prostřední část tak, že pokud je počet nepotřebných vzorků lichý, zvolíme střed užitečného signálu blíže počátku.
- Výsledek uložíme do vektoru $a^{(j-1)}$, který slouží jako vstupní pro další krok rekonstrukce
- Snížíme j o jedničku. Pokud je nyní j = 1 algoritmus končí a vektor $a^{(1)}$ se uloží jako výstupní signál.

Poznámky k algoritmu:

- po skončení algoritmu máme ze vstupních koeficientů ve vektorech $a^{(d)}, d^{(d)}, d^{(d-1)}, d^{(d-2)}, \ldots..,\ d^{(1)}$ vytvořený výstupní signál
- ad 1. v prvním kroku algoritmu tedy nadvzorkujeme $a^{(j)}$ na délku $2n$
- ad 2. délka výsledné konvoluce v prvním kroku je $(2n+m)$
- ad 3. sečtený vektor má tedy délku $(2n+m)$
- ad 4. touto operací dostáváme vektor délky shodné s délkou původního signálu.

3.2 Realizace

Základem externího programu je funkce myWavelet, která v sobě obsahuje algoritmy DTWT a IDTWT. Jejím vstupem je vektor navzorkovaného vstupního signálu přivedeného do levého inletu a výstupem opět pole vektorů stejné délky jako pole vstupního vektoru. Vzorem pro tuto funkci mi byla literatura [5]. Funkce je vložena do metody wavelet_perform(). Funkce _perform je „srdcem DSP“ každé třídy pro práci se signálem, co je to DSP metoda je popsáno v kapitole 2.6. Celý zdrojový kód externího programu je opět vybaven poznámkami a dostupný na přiloženém CD.

Bohužel kvůli problému s nastavením překladače se mi nepodařilo vytvořit funkční knihovnu. Při stejném nastavení, jaké jsem použil pro vzorový příklad ahoj, se sice vytvoří dynamická knihovna, ale PD si ji nenačte. Tento problém se mi projevoval u každého externího programu, který pracoval se signálem. Po poradě s vedoucím jsem tedy na přiložené CD přidal verzi funkce pro výpočet waveletové transformace, která si vstupní hodnoty načítá z textového souboru s názvem input, a výstupní hodnoty ukládá do textového souboru s názvem output. V souboru testovani na přiloženém CD je soubor myWavelet.exe, který po spuštění načte hodnoty z textového souboru input.txt. Pod těmito hodnotami si můžeme představit pole vzorků vstupního signálu. Po načtení se provede výpočet DTWT a IDTWT. Výsledek se uloží do textového souboru

s názvem output.txt. Pod těmito hodnotami si můžeme představit vzorky výstupního signálu.

Oproti funkci, která by byla použita pro realizaci v PD, obsahuje navíc část mainWavelet.cpp. Ta se stará o práci se soubory a taky je v ní počítána délka vstupního vektoru. Toto by ve funkci pro PD bylo pevně nastaveno. Protože PD pracuje se segmenty o velikosti 64 vzorků. Veškeré parametry pro výpočet waveletové transformace, jako je například hloubka dekompozice nebo koeficienty použitých filtrů, jsou nastavitelné v hlavičkovém souboru constWavelet.h.

Hlavní část celé funkce je uložena v souboru myWavelet.cpp. V tomto souboru je naprogramován výpočet konečné diskrétní waveletové transformace a zpětné konečné diskrétní waveletové transformace. Kód je vybaven poznámkami, které vysvětlují význam jednotlivých funkcí a použitých proměnných.

ZÁVĚR

V kapitole 1 je uvedeno teoretické minimum problematiky waveletové transformace. Uvedl jsem v ní základní matematické vzorce pro výpočet transformace. V poslední podkapitole je uveden mallatův pyramidový algoritmus. Díky tomuto algoritmu se z waveletové transformace stal z matematických teorií skutečný nástroj pro zpracování signálů.

V kapitole 2 jsem popsal základní strukturu prostředí Pure Data, vyvinutého Millerem Puckettem. Jedná se o prostředí pro zpracování hudebního signálu a prioritně pro tvorbu vlastní hudby. V této kapitole je uveden základní popis jednotlivých schránek, jejich vzhled a význam. Jsou zde též dva jednoduché patche pro ukázku, jak prostředí PD vypadá a jak se chová. Z nich je patrná specifická „schránkovitá“ struktura patchů v prostředí Pure data. Po seznámení se s PD a prostudování manuálu, jsem dospěl k názoru, že waveletovou transformaci nebude prvky PD možno realizovat. Proto je v kapitole 2.5 uveden postup jak se vytváří vlastní objekty použitelné v PD. Na velice jednoduchém příkladu jsou popisovány jednotlivé části kódu. Pro realizaci transformace však bylo nutno pracovat se signálovými objekty, proto jsem v kapitole 2.6 popsal jak definovat v kódu třídu pro práci se signálem.

Poslední kapitola je věnována problému realizace waveletové transformace v prostředí Pure Data. Po tvorbě externích programů pro PD jsem se v práci zaměřil na výběr vhodného algoritmu řešení. Z důvodů uvedených v úvodu kapitoly 3 jsem zvolil algoritmus konečné diskrétní waveletové transformace (DTWT) a inverzní konečné diskrétní waveletové transformace (IDTWT). Oba tyto algoritmy jsem ve své práci popsal. Vlastní realizace waveletové transformace v prostředí PD spočívala ve vytvoření funkce, která by realizovala DTWT a IDTWT, a byla začleněna do kódu pro vytvoření signálového objektu v PD.

SEZNAM POUŽITÉ LITERATURY

[1] HUBBARD, B. B.: The world according to wavelets: the story of matematical technique in the making. 2.vydání. Wellesley,Massachusetts: A K Peters, 1998. ISBN 156881-0725.

[2] PUCKETTE, M.: Theory and Techniques of Electronic Music. World Scientific Publishing Company (May 23, 2007). ISBN 9812700773.

[3] PUCKETTE, M.: Pd Documentation. [on-line] . dostupné z: http://crca.ucsd.edu/~msp/

[4] RAJMIC, P.: Využití waveletové transformace a matematické statistiky pro separaci signálu a šumu. PhD Thesis, Brno university of Technology, Brno, 2004.

[5] SOUKAL, P.: Algoritmy pro výpočet waveletové transformace v reálném čase, jejich analýza a optimalizace. Diplom Thesis, Brno university of Technology, Brno, 2006.

[6] VARGIC, R.: Wavelety a banky filtrov. STU, Bratislava, 2004. ISBN 8022720933

[7] WICKERHAUSER, M. V. : Adapted Wavelet Analysis from Theory to Software. A K Peters, 1993. ISBN 1-56881-041-5

SEZNAM PŘÍLOH

Příloha č.1

Nastavení Microsoft Visual Studia 2005 na vytvoření dll knihovny pro PD

Pro vytvoření dynamické knihovny použitelné pro PD je třeba udělat následující kroky :

- Přidat k projektu knihovnu m_pd.h (tu lze nalézt na přiloženém CD)
- Napsat zdrojový kód externího programu v jazyce C a přidat ho k projektu
- Zpřístupnit knihovnu PD: Zkopírujeme ze složky pd/bin/ knihovnu z názvem pd.lib do adresáře C:/Windows/Temp/
- Nastavit konfigurační vlastnosti projektu :
  a) Nastavit konfigurační typ na .dll.
     Properties->Configuration Properties->General a v řádku Configuration Type změnit na Dynamic Library(.dll)
  b) Přidat MSW do definic preprocesoru.
     Properties->Configuration Properties->C/C++->Preprocesor a do pole Preprocesor Definition vepsat MSW
  c) Říci compileru jaký má použít jazyk
     Properties->Configuration Properties-> C/C++->Advanced a v řádku Compile As vybrat možnost Compile as C Code (/TC)
  d) Říci linkeru kde najít pd.lib
     Properties->Configuration Properties->Linker->Input a do řádku Additional Dependencies napsat C:/Windows/Temp/pd.lib
  e) Říci linkeru aby exportoval funkci setup
     Properties->Configuration Properties->Linker->Comand Line a do kolonky Additional options napsat /export:mujprogram_setup

Po těchto krocích stačí kliknout na tlačítko build solution a vytvoří se dll knihovna použitelná pro PD.

Příloha č.2

Instalace PD a vytvoření a spuštění patche

Instalace

Nejdříve je třeba si nainstalovat program Pure Data. Instalační soubor s nejnovějšími verzemi lze nalézt na stránce http://crca.ucsd.edu/~msp/software nebo je možné stáhnout si exteded verze, které naleznete na stránce http://puredata.info/downloads. Tyto extended verze obsahují daleko více objektů pro vytváření patchů. Navíc jsou většinou koncipovány jako samoinstalační program, ten nám po spuštění bez obtíží nainstaluje program Pure Data.

Vytvoření vlastního patche

Spustíme program Pure Data. Jako první nám naběhne hlavní okno PD. Pro vytvoření vlastního nového patche klikneme na lištu File a vybereme New. Po tomto kroku se otevře druhé okno s prázdným patchem. Jednotlivé schránky vkládáme kliknutím na lištu Put a poté na danou schránku. Po vytvoření a propojení jednotlivých schránek patche můžeme přejít ke spuštění.

Spuštění patche

Je třeba se přepnout do editačního módu, v záložce Edit kliknout na Edit mode. V tomto módu už by měli pracovat všechny prvky. Ale pokud pracujeme se zvukovými signály, je třeba ještě zapnout audio. Kliknutím na záložku Media a v ní kliknout na audio ON. Tento postup je třeba dodržet i při otvírání již vytvořených patchů.

Příloha č.3

Obsah přiloženého CD

Přiložené CD obsahuje adresáře :

/zdrojove kody

Obsahuje zdrojové kódy externích programů pro PD a zdrojový kód testovací verze funkce myWavelet.

/patche

Zde jsou dostupné všechny patche vzorových příkladů pro PD uvedené v této práci.

/knihovna m_pd.h

Obsahuje knihovnu m_pd.h.

/dll knihovny

Obsahuje dll knihovny externích programů.

/testovani

Obsahuje testovací verzi funkce pro výpočet waveletové transformace.